# Secure input handling
# -
# insights from the last decade

**Erik Poll**

**Digital Security**

**Radboud University, Nijmegen, the Netherlands**

# 10 years ago

**Sergey Bratus & Meredith Patterson present LangSec at CCC 2012**

- **LangSec = Language-Theoretic Security**

**Highlighting the role of input languages in security**

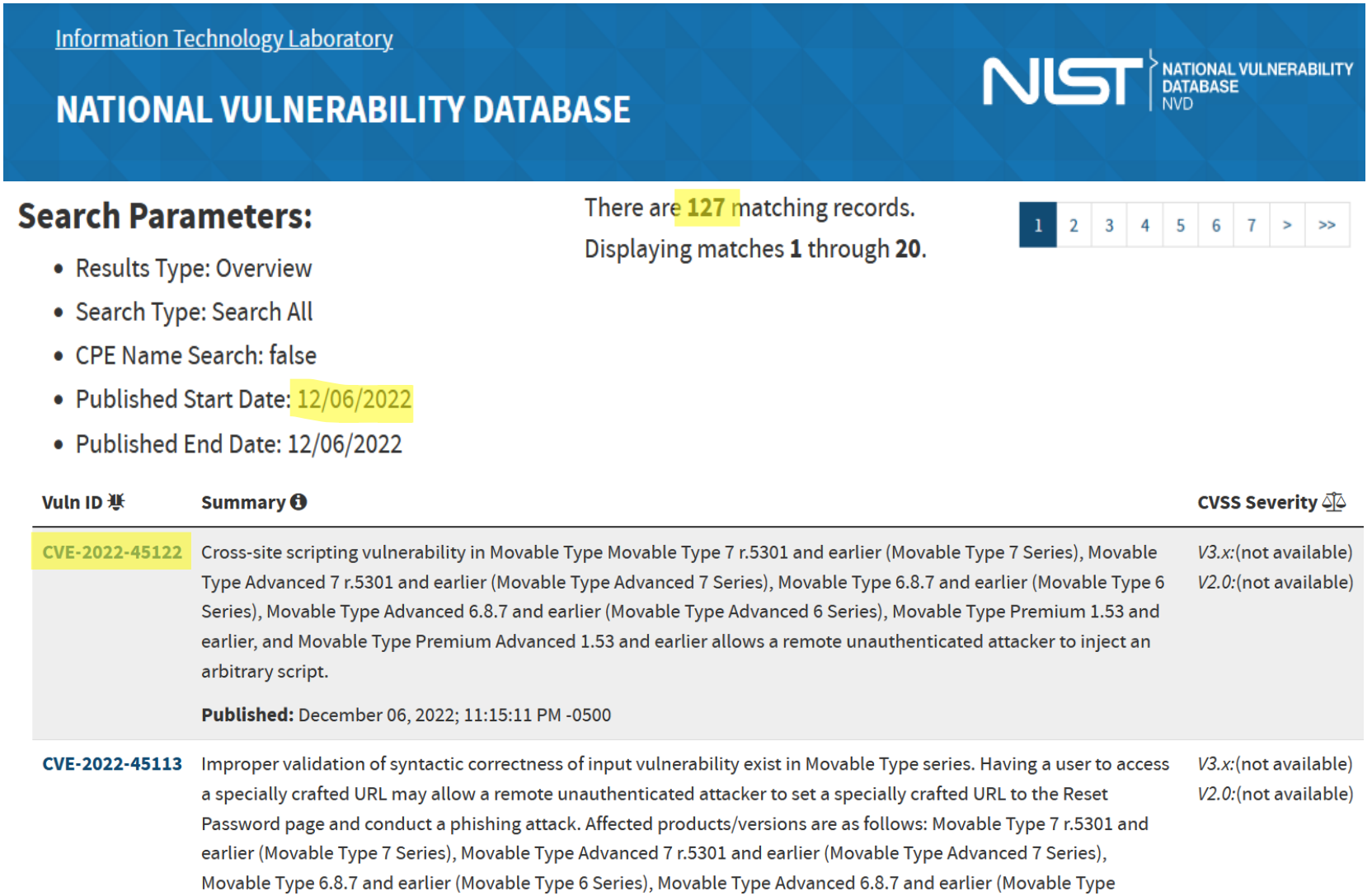

**'The science of insecurity'**

**http://www.youtube.com/watch?v=3kEfedtQVOY**


# Software is *the* root cause of security problems

## Things can be hacked *because* (and if?) there is software in them

**Last Tuesday: 127 CVEs. This year > 45,000.**



Information Technology Laboratory

NATIONAL VULNERABILITY DATABASE

NIST NATIONAL VULNERABILITY DATABASE NVD

**Search Parameters:**

- Results Type: Overview
- Search Type: Search All
- CPE Name Search: false
- Published Start Date: 12/06/2022
- Published End Date: 12/06/2022

There are 127 matching records.
Displaying matches **1** through **20**.

1 2 3 4 5 6 7 > >>

| Vuln ID | Summary | CVSS Severity |
|---|---|---|
| CVE-2022-45122 | Cross-site scripting vulnerability in Movable Type Movable Type 7 r.5301 and earlier (Movable Type 7 Series), Movable Type Advanced 7 r.5301 and earlier (Movable Type Advanced 7 Series), Movable Type 6.8.7 and earlier (Movable Type 6 Series), Movable Type Advanced 6.8.7 and earlier (Movable Type Advanced 6 Series), Movable Type Premium 1.53 and earlier, and Movable Type Premium Advanced 1.53 and earlier allows a remote unauthenticated attacker to inject an arbitrary script.<br>**Published:** December 06, 2022; 11:15:11 PM -0500 | *V3.x:*(not available)<br>*V2.0:*(not available) |
| CVE-2022-45113 | Improper validation of syntactic correctness of input vulnerability exist in Movable Type series. Having a user to access a specially crafted URL may allow a remote unauthenticated attacker to set a specially crafted URL to the Reset Password page and conduct a phishing attack. Affected products/versions are as follows: Movable Type 7 r.5301 and earlier (Movable Type 7 Series), Movable Type Advanced 7 r.5301 and earlier (Movable Type Advanced 7 Series), Movable Type 6.8.7 and earlier (Movable Type 6 Series), Movable Type Advanced 6.8.7 and earlier (Movable Type | *V3.x:*(not available)<br>*V2.0:*(not available) |

# Software engineers don't understand security!

# What we tell software engineers:

## OWASP Top 10 [2017]

1. Injection
2. Broken Authentication
3. Sensitive Data Exposure
4. XML External Entities (XXE)
5. Broken Access Control
6. Security Misconfiguration
7. Cross-Site Scripting (XSS)
8. Insecure Deserialization
9. Using Components with Known Vulnerabilities
10. Insufficient Logging & Monitoring

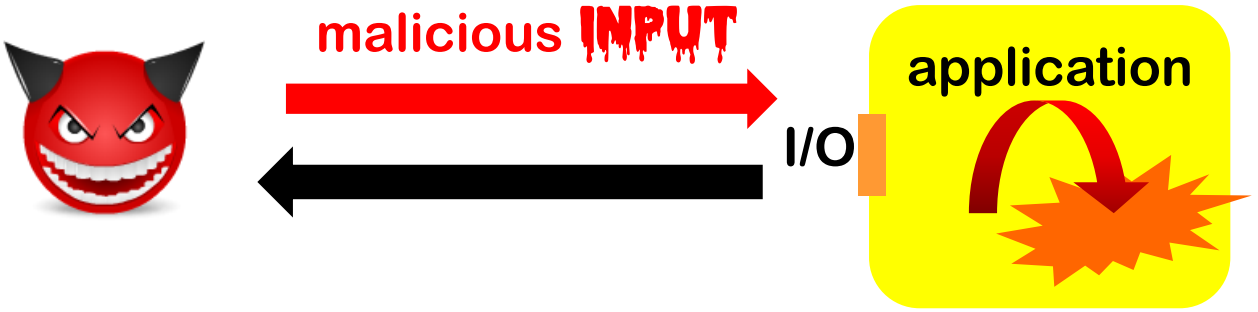
## CWE TOP 25 [2022]

1 Out-of-bounds Write
2 Cross-site Scripting
3 SQL Injection
4 Improper Input Validation
5 Out-of-bounds Read
6 OS Command Injection
7 Use After Free
8 Path Traversal
9 Cross-Site Request Forgery (CSRF)
10 Unrestricted Upload of File with Dangerous Type
11 NULL Pointer Dereference
12 Deserialization of Untrusted Data
13 Integer Overflow or Wraparound
14 Improper Authentication
15 Use of Hard-coded Credentials
16 Missing Authorization
17 Command Injection
18 Missing Authentication for Critical Function
19 Improper Restriction of Bounds of Memory Buffer
20 Incorrect Default Permissions
21 Server-Side Request Forgery (SSRF)
22 Race Condition
23 Uncontrolled Resource Consumption
24 Improper Restriction of XML External Entity Reference
25 Code Injection

## CWE TOP 1000

**We do not make it to easy enough for software engineers to get security right!**

# There's only ONE main problem: INPUT handling




Garbage In, Garbage Out

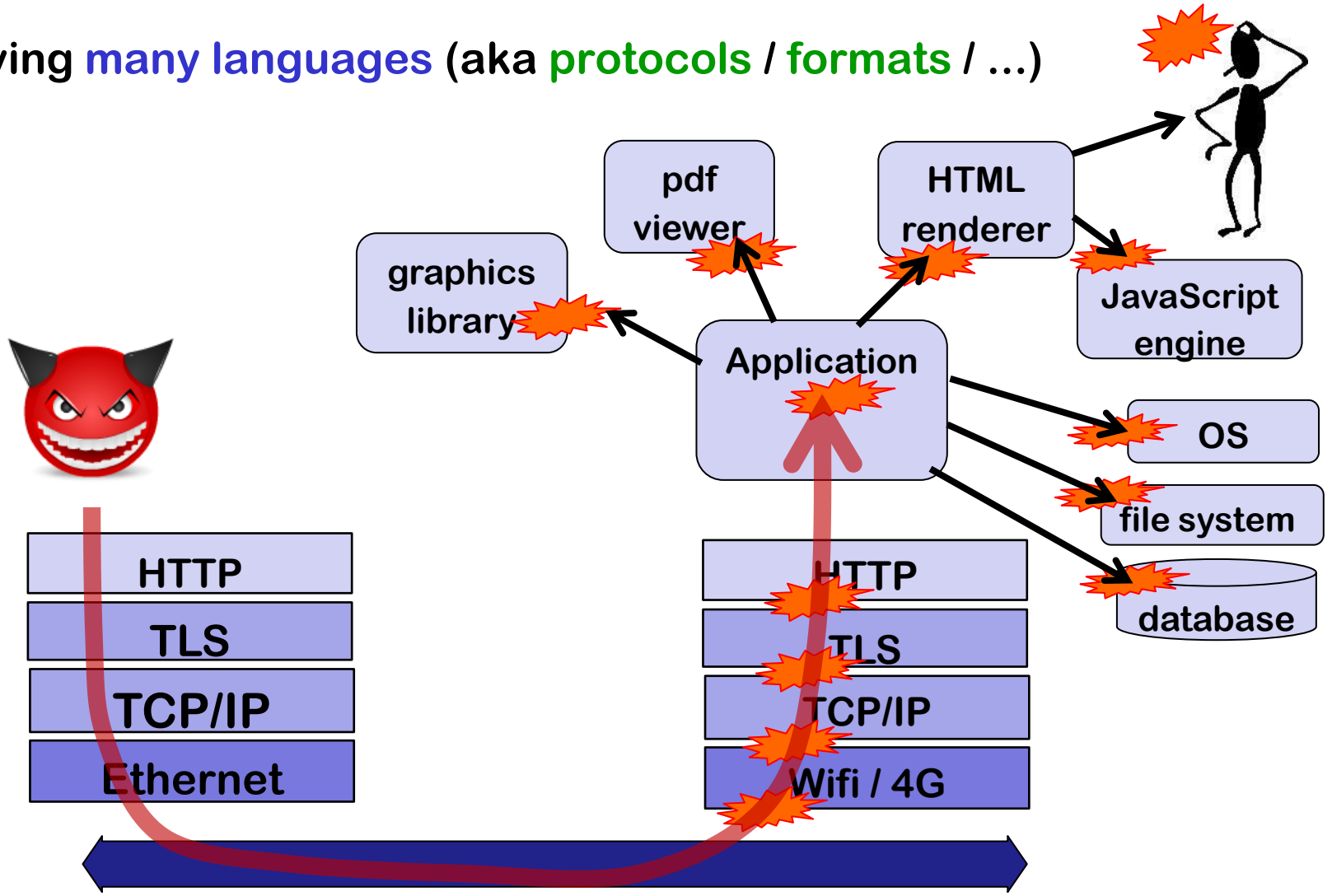
becomes *Malicious* Garbage In, *Security Incident* Out

or Garbage In, Evil Out

# INPUT problems are PARSING problems

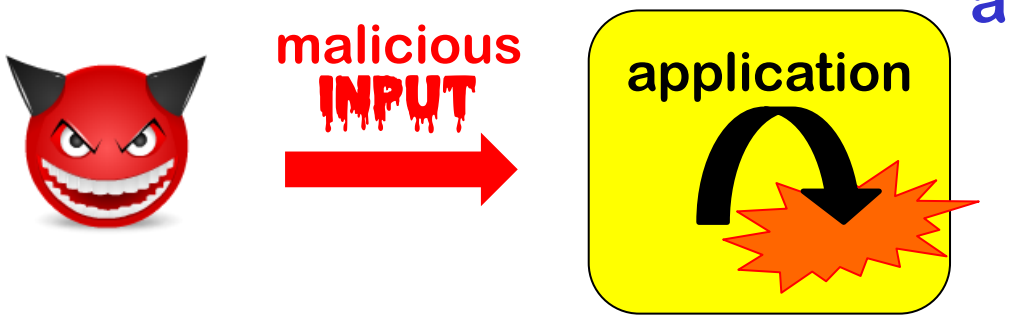
Input is parsed (aka decoded / interpreted/...) in many places.

Involving many languages (aka protocols / formats / ...)

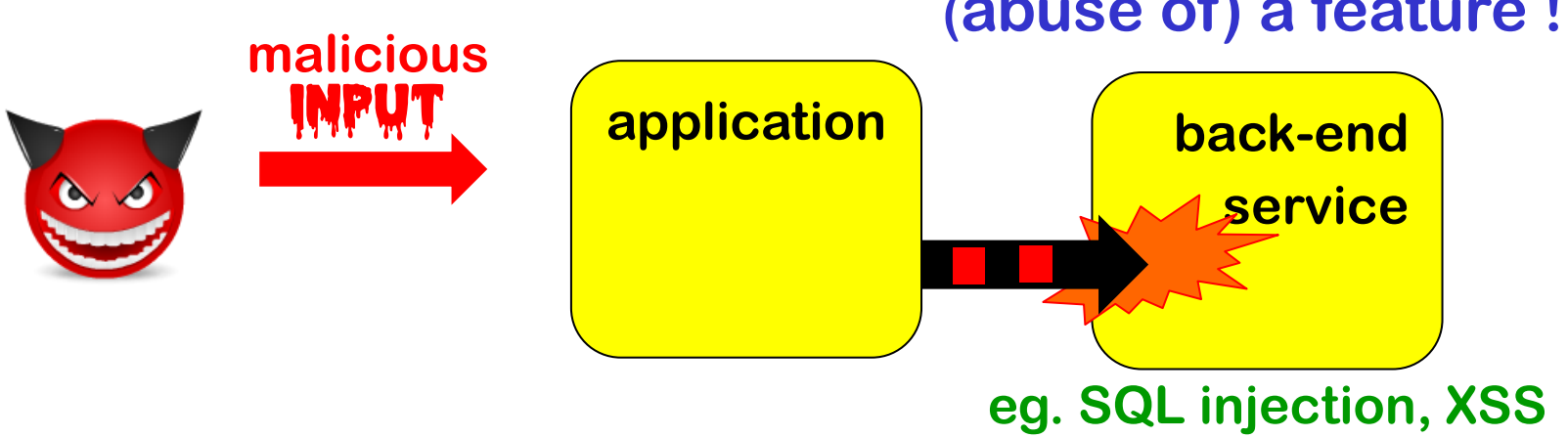


# Two types of input flaws: bugs & features

## 1. Processing Flaws

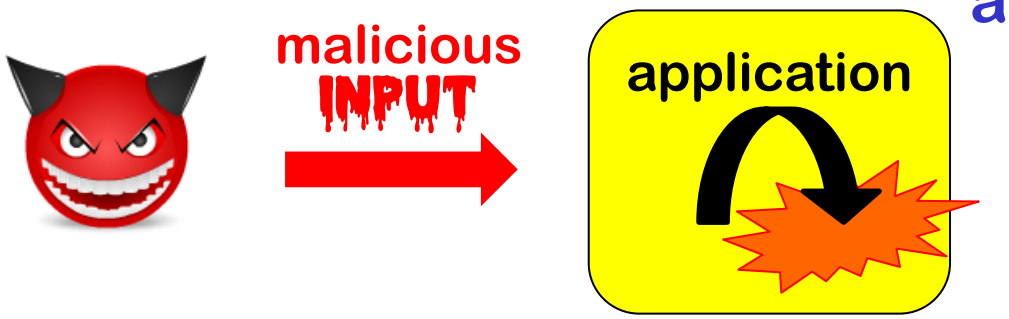



## 2. Injection Flaws

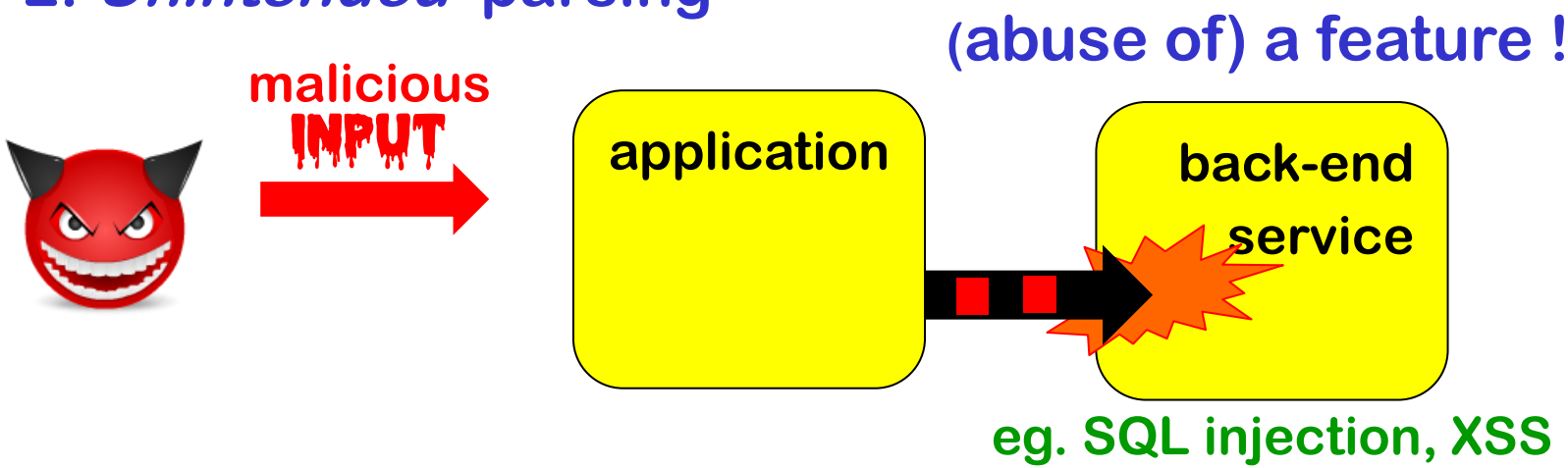


# In terms of parsing: buggy & unintended parsing

## 1. *Insecure, buggy* parsing




## 2. *Unintended* parsing



# Tackling buggy parsing:

# using the LangSec approach

# Example security flaws due to buggy parsing

CVE-2022-43667

Stack-based *buffer overflow* vulnerability in CX-Programmer may lead to *information disclosure* and/or *arbitrary code execution* by having a user to open a specially crafted CXP file.

Published: December 06, 2022; 11:15:10

CVE-2022-41325

An *integer overflow* in VideoLAN VLC Media Player allows attackers, by tricking a user into opening a crafted playlist or connecting to a rogue VNC server, to *crash* VLC or *execute code* ...

Published: December 06, 2022; 11:15:11

CVE-2022-40918

*Buffer overflow* in in Force 1 Discovery U818A HD+ FPV Drone allows attacker to gain *remote code execution* as root via a specially crafted UDP packet.

Published: December 06, 2022; 7:15:10

# Root causes of buggy parsing

1. Many input languages / formats:

   CXP, VLC playlist, VNC/VLC format, UDP packet,

   Wifi, Ethernet, Bluetooth, GSM/3G, 4G, 5G, ...

   HTTP(S), TLS, SSH, OpenVPN, ...

   URLs, X509 certificates, domain names, ...

   JPG, MP3, MPEG, ...

   HTML, PDF, Word, Excel, Powerpoint

2. Often these are complex and/or poorly specified

3. Hand-written parser code, often in unsafe languages like C(++)

Fuzzing – aka fuzz testing – is a great way to find these bugs!

# LangSec: tackling buggy parsing

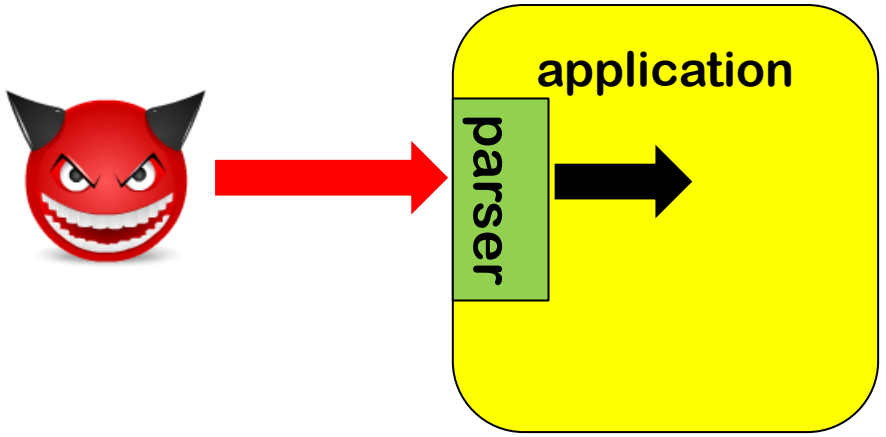
**Solution**

1. **Provide clear, formal spec of input language**

   eg as regular expression or BNF grammar

2. **Generate parser code**

   using a parser generator tool




For more: see langsec.org

# Tackling unintended parsing (ie injection attacks)

# use types!

[Strings considered harmful , USENIX :login; 2019]

# Example unintended parsing – ie injection flaws

- CVE-2022-45217
  Cross-site scripting (XSS) in Book Store Management System allows attackers to execute arbitrary web scripts or HTML via a crafted payload injected into the Level parameter under the Add New System User module.

  Published: December 06, 2022; 9:15:10

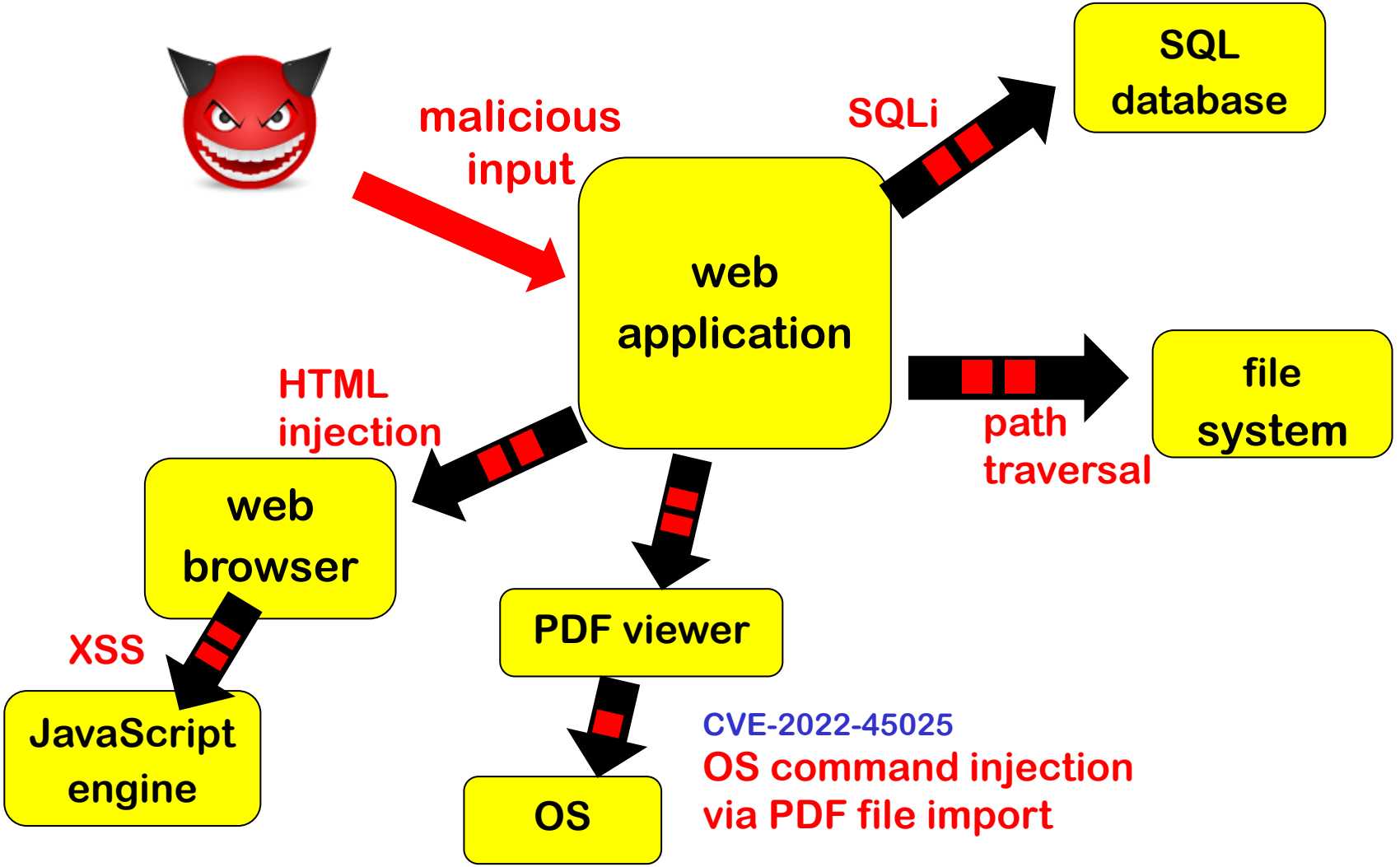
- CVE-2022-33875
  SQL Injection vulnerability in Fortinet FortiADC allows an attacker to execute unauthorized code or commands via specifically crafted HTTP requests.

  Published: December 06, 2022; 12:15:10

- CVE-2022-45025
  Markdown Preview Enhanced for VSCode and Atom contains a command injection vulnerability via the PDF file import function.

  Published: December 06, 2022; 9:15:10

# Many back-ends, with input languages, more problems with unintended parsing …



# Root causes of unintended parsing

1. Many languages: e.g HTML, SQL, PDF, OS commands
   - Also as output language as well as input languages
   - Combined in complex way, e.g OS commands inside PDF (?)
2. Complex data flows where user input can end up being interpreted as one of these languages
3. Very powerful, expressive languages

   JavaScript in HTML, JavaScript or ActionScript in PDF,
   SQL commands, OS commands, ...

# Anti-pattern: STRINGS

**Strings are *useful*, because you use them to represent many things:**

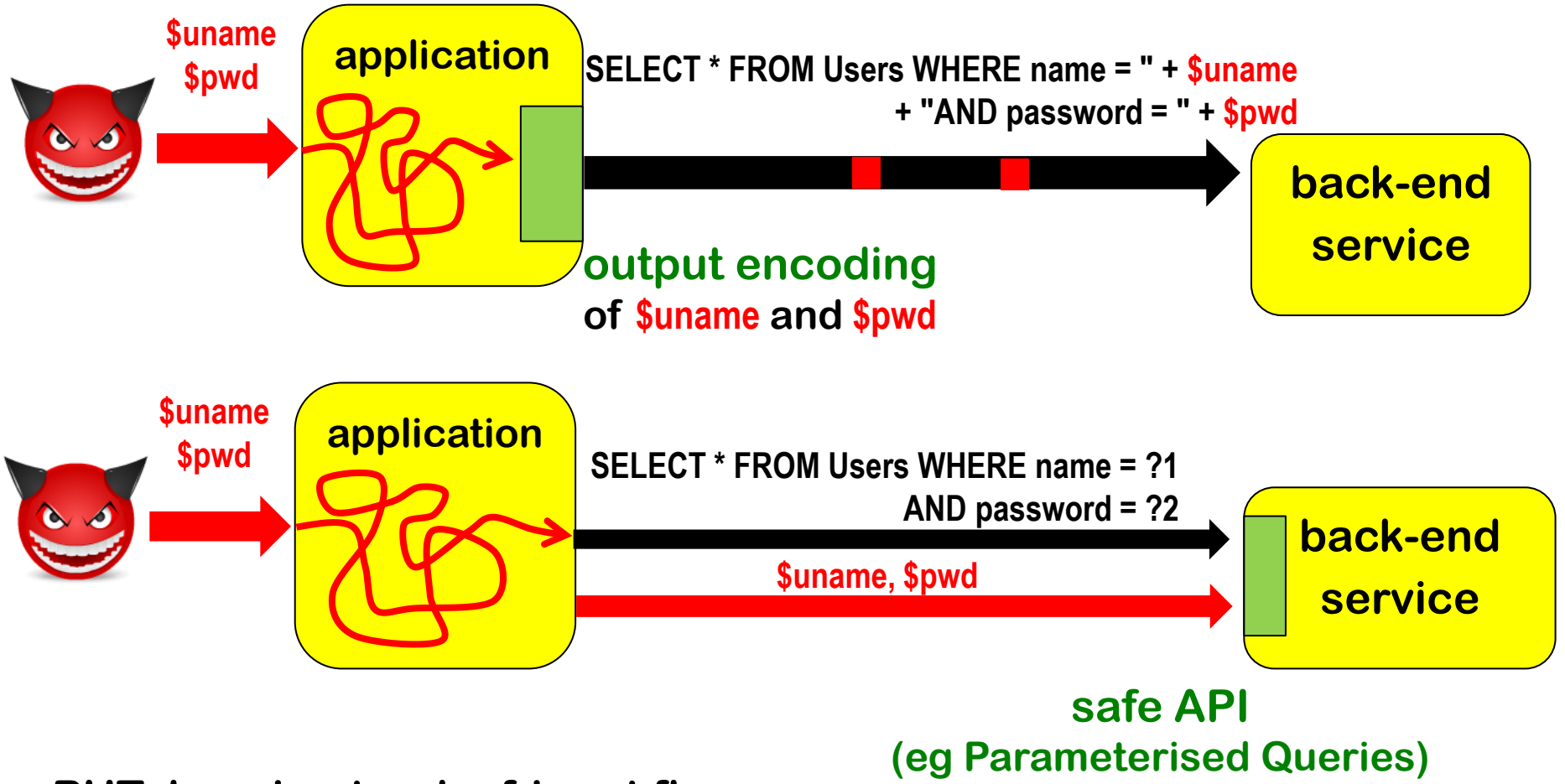
**eg. user name, file name, email address, URL, shell command, snippet of SQL, HTML, or JavaScript, ...**

- **Not just `String` but also `char*`, `char[]`, `StringBuilder`, ...**

**This also make strings *dangerous:***

1. **A string is unstructured & unparsed data, and processing it often involves some interpretation - incl. parsing**
2. **The same string may be handled & interpreted in many – possibly unexpected – ways**
3. **A string parameter in an API call can – and often does – hide a very expressive & powerful language**

# Solutions: output encoding or safe APIs



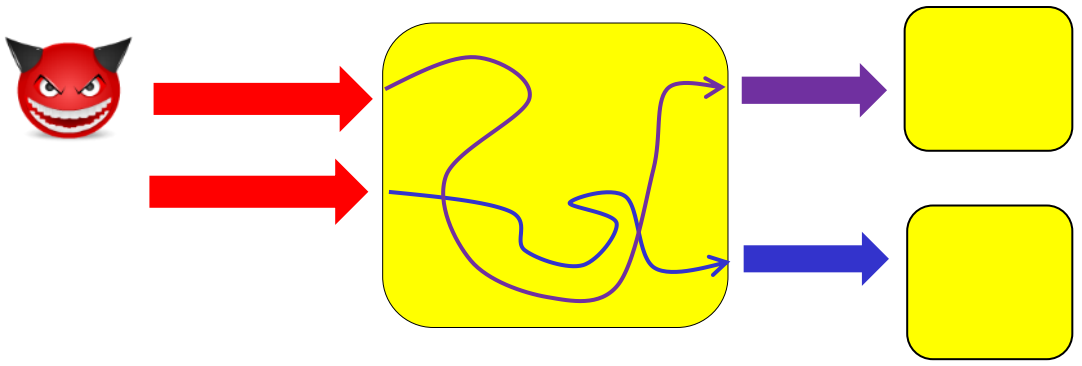

**BUT: keeping track of input flows through the application remains a nightmare!**

# Remedy: Types (1) to distinguish *languages*

**Instead of using strings for everything,**
**use different types to distinguish different kinds of data**

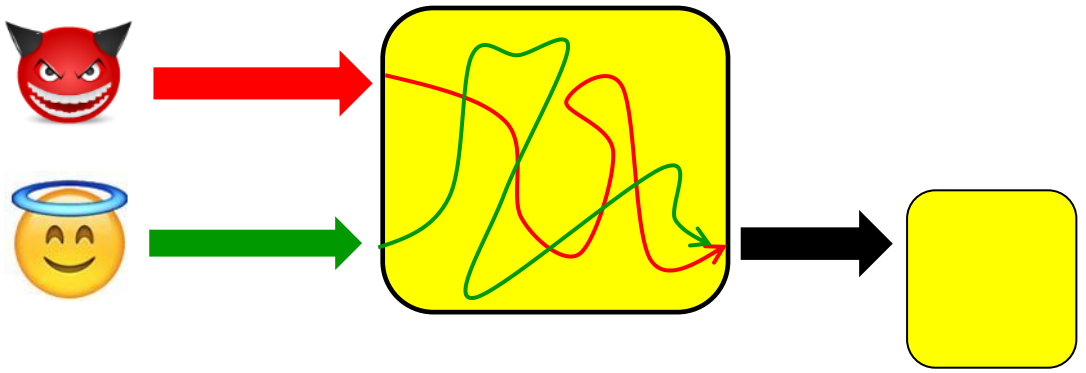
> **Eg different string-like wrapper types for HTML, URLs, file names, user names, paths, …**

- **Advantage: no ambiguity about the intended use of data**



# Remedy: Types (2) to distinguish *trust levels*

Use types to track origin and control destination of data



- Eg trusted HTML that contains JavaScript we're happy to execute vs untrusted HTML that needs validation or encoding before it reaches a browser engine
- Typical distinction: user input vs compile-time constants

# Example: Trusted Types DOM API in Chrome browser

Trusted Types initiative to root out DOM-based XSS

replaces string-based DOM API with typed API

- Type checking ensures that untrusted data can only reach dangerous APIs after passing (carefully vetted) validation or encoding operations

```
TrustedHTML htmlEncode(String str)
TrustedHTML create(@Compiletimeconstant String str)
```

[https://github.com/WICG/trusted-types]

# Conclusions

- Most security flaws are **INPUT** processing flaws
- These flaws involve **PARSING** one of many input languages / formats
- LangSec provides a way to tackle **BUGGY PARSING**
  - by generating parser code from unambiguous, formal spec
- Using types (and avoiding the use of **STRINGS**) we can prevent **UNINTENDED PARSING** parsing – and so-called injection attacks
  - using types to distinguish languages / formats and trust levels

# Further reading/watching

- On LangSec:
  - Sergey Bratus & Meredith Patterson, The science of insecurity, CCC 2012, http://www.youtube.com/watch?v=3kEfedtQVOY
  - Much more on langsec.org
- On avoiding strings and using (trusted) types
  - Christoph Kern, Preventing Security Bugs through Software Design, AppSec California 2016, https://www.youtube.com/watch?v=ccfEu-Jj0as
  - Wang et al., If It's Not Secure, It Should Not Compile: Preventing DOM-Based XSS in Large-Scale Web Development with API Hardening, ICSE'2021
  - Erik Poll, Strings considered harmful , USENIX :login; , 2019
- Or, if you have more time, read my lecture notes on Secure Input Handling